Medical Chatbot: Enhancing PhoBert with Domain-Specific Knowledge

System design

Patient ↔️ Chatbot ↔️ NLP Engine ↔️ Diagnosis Model ↔️ Triage System

↘️↗️

Database

2.1. Chatbot Interface (Frontend + NLP Middleware)

Công nghệ: React / Flutter / Web-based chat + FastAPI backend

Xử lý tiếng nói / văn bản: sử dụng Whisper (voice-to-text) và TTS (Text-to-Speech, nếu cần)

NLP Middleware:

Fine-tuned LLM (e.g., LLaMA-3, Mistral, GPT-4) cho hội thoại mềm dẻo

Intent Detection: phân loại mục đích câu hỏi (symptom reporting, greeting, urgency, etc.)

Slot filling: trích xuất các thông tin như tuổi, giới tính, triệu chứng, thời gian khởi phát...

2.2. Thông tin bệnh nhân & lưu trữ

Dữ liệu trích xuất:

json

Copy

Edit

{

"name": "Nguyen Van A",

"age": 45,

"gender": "male",

"symptoms": ["ho", "sốt", "khó thở"],

"onset": "3 ngày trước",

"risk\_factors": ["cao huyết áp", "tiểu đường"]

}

Ghi vào PostgreSQL/MySQL/MongoDB

Giao diện quản trị bác sĩ có thể xem và chỉnh sửa

2.3. Mô hình dự đoán bệnh

Input features:

Symptom embeddings (dùng BERT/PhoBERT)

Metadata (tuổi, giới tính, bệnh nền, thời gian phát bệnh)

Model architecture:

Multi-modal deep network (NLP + tabular data)

Loss function: multi-label classification (vì có thể nhiều bệnh)

Output:

{ "probability": {"viêm phổi": 0.83, "covid-19": 0.65, "cảm lạnh": 0.21} }

Chọn top-1 hoặc top-k làm dự đoán ban đầu

2.4. Phân tuyến (Triage Decision)

Rule-based (ban đầu): kết hợp độ tin cậy của model + bộ luật y tế

Gợi ý tuyến:

"Nội hô hấp", "Khám sàng lọc COVID", "Cấp cứu", ...

Sau này có thể huấn luyện mô hình phân loại tuyến nâng cao bằng supervised learning.

Pipeline hoạt động

Bệnh nhân vào giao diện chatbot

Chatbot hỏi các câu hỏi dẫn dắt:

"Chào bạn, bạn cảm thấy không khỏe ở điểm nào?"

"Triệu chứng xuất hiện từ bao giờ? Có bệnh nền không?"

NLP engine trích xuất thông tin quan trọng → lưu vào database

Mô hình dự đoán bệnh sử dụng thông tin → trả kết quả

Hệ thống quyết định phân tuyến dựa trên kết quả và độ ưu tiên

Trả lời lại bệnh nhân:

"Bạn có thể đang bị viêm phổi. Tôi khuyên bạn nên đến khoa Nội hô hấp để được kiểm tra thêm."

Kết thúc bằng tạo record trong DB

**Training & Deployment**

**Dữ liệu:**

* EMR lịch sử, báo cáo bệnh nhân
* Annotation thủ công: gán label bệnh chính xác (nếu cần)
* Augmentation: paraphrase triệu chứng, mô phỏng hội thoại

**Training:**

* Framework: PyTorch (khuyến nghị sử dụng torchtext hoặc transformers)
* Sử dụng Lightning hoặc HuggingFace Trainer để tổ chức training pipeline
* Early stopping, validation loss monitoring

**Deployment:**

* API inference backend (FastAPI + Uvicorn/Gunicorn)
* GPU model server (e.g., Triton Inference Server hoặc TorchServe)
* DB kết nối qua SQLAlchemy / ORM
* Giao tiếp frontend dùng REST hoặc WebSocket

**Mô hình hội thoại mềm dẻo**

* Fine-tune LLM với **Medical Chat Dataset** (kết hợp data từ MedDialog, iCliniq, HealthTap)
* Prompt Engineering (ví dụ):

You are a medical assistant helping to triage patients. You should speak politely, empathetically, and gather necessary symptoms to support a medical triage decision. Ask 2-3 follow-up questions if needed.

#Patient Triage Chatbot

from fastapi import FastAPI, Request, Depends

from pydantic import BaseModel

from typing import List, Dict, Optional

import uvicorn

import random

import uuid

import torch

import torch.nn as nn

import torch.nn.functional as F

from transformers import AutoTokenizer, AutoModel

from sqlalchemy import create\_engine, Column, String, Float, Text

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker, Session

import json

app = FastAPI()

# ======= Database Setup ========

DATABASE\_URL = "mysql+pymysql://username:password@localhost:3306/triage\_db"

engine = create\_engine(DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative\_base()

class PatientRecord(Base):

\_\_tablename\_\_ = "patient\_records"

record\_id = Column(String(64), primary\_key=True)

user\_id = Column(String(64))

symptoms = Column(Text)

predicted\_diseases = Column(Text)

recommended\_department = Column(String(128))

chat = Column(Text)

Base.metadata.create\_all(bind=engine)

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

# ======= Input/Output Models ========

class ChatInput(BaseModel):

user\_id: str

message: str

class ChatOutput(BaseModel):

response: str

record\_id: Optional[str]

predicted\_diseases: Optional[Dict[str, float]]

recommended\_department: Optional[str]

conversation\_store = {}

# ======= Symptom Classifier Model ========

class SymptomClassifier(nn.Module):

def \_\_init\_\_(self, num\_labels=5):

super(SymptomClassifier, self).\_\_init\_\_()

self.bert = AutoModel.from\_pretrained("vinai/phobert-base")

self.dropout = nn.Dropout(0.3)

self.fc = nn.Linear(self.bert.config.hidden\_size, num\_labels)

def forward(self, input\_ids, attention\_mask):

outputs = self.bert(input\_ids=input\_ids, attention\_mask=attention\_mask)

pooled\_output = outputs[1] if len(outputs) > 1 else outputs[0][:, 0]

x = self.dropout(pooled\_output)

return torch.sigmoid(self.fc(x))

# ======= Load Model for Inference ========

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

tokenizer = AutoTokenizer.from\_pretrained("vinai/phobert-base")

label\_map = {

0: "viêm phổi",

1: "covid-19",

2: "thiếu máu",

3: "cao huyết áp",

4: "ngộ độc thực phẩm"

}

model = SymptomClassifier(num\_labels=5).to(device)

model.load\_state\_dict(torch.load("model\_weights.pt", map\_location=device))

model.eval()

# ======= Helper Functions ========

def extract\_symptoms(message: str) -> List[str]:

keywords = ["ho", "sốt", "khó thở", "đau đầu", "chóng mặt", "đau bụng", "tiêu chảy"]

return [kw for kw in keywords if kw in message.lower()]

def predict\_disease\_with\_model(message: str) -> Dict[str, float]:

inputs = tokenizer(message, return\_tensors="pt", truncation=True, padding=True, max\_length=128)

inputs = {k: v.to(device) for k, v in inputs.items()}

with torch.no\_grad():

logits = model(\*\*inputs)

probs = logits.squeeze().tolist()

return {label\_map[i]: round(float(probs[i]), 2) for i in range(len(probs)) if probs[i] > 0.5}

def recommend\_department(disease: str) -> str:

mapping = {

"viêm phổi": "Nội hô hấp",

"covid-19": "Khám sàng lọc COVID",

"thiếu máu": "Huyết học",

"cao huyết áp": "Tim mạch",

"ngộ độc thực phẩm": "Tiêu hóa"

}

return mapping.get(disease, "Khám tổng quát")

@app.post("/chat", response\_model=ChatOutput)

async def chat(input: ChatInput, db: Session = Depends(get\_db)):

user\_id = input.user\_id

message = input.message

symptoms = extract\_symptoms(message)

conv = conversation\_store.setdefault(user\_id, [])

conv.append({"user": message})

if symptoms:

predicted = predict\_disease\_with\_model(message)

if not predicted:

predicted = {"bệnh không xác định": 0.5}

dept = "Khám tổng quát"

top\_disease = "bệnh không xác định"

else:

top\_disease = max(predicted, key=predicted.get)

dept = recommend\_department(top\_disease)

record\_id = str(uuid.uuid4())

# Lưu vào cơ sở dữ liệu

record = PatientRecord(

record\_id=record\_id,

user\_id=user\_id,

symptoms=json.dumps(symptoms, ensure\_ascii=False),

predicted\_diseases=json.dumps(predicted, ensure\_ascii=False),

recommended\_department=dept,

chat=json.dumps(conv, ensure\_ascii=False)

)

db.add(record)

db.commit()

response = f"Tôi ghi nhận bạn có các triệu chứng: {', '.join(symptoms)}. Dự đoán bạn có thể bị '{top\_disease}'. Bạn nên đến khoa '{dept}' để được khám thêm."

return ChatOutput(

response=response,

record\_id=record\_id,

predicted\_diseases=predicted,

recommended\_department=dept

)

else:

response = "Bạn có thể cho tôi biết rõ hơn về triệu chứng bạn đang gặp phải không?"

return ChatOutput(response=response)

if \_\_name\_\_ == "\_\_main\_\_":

uvicorn.run(app, host="0.0.0.0", port=8000)

from fastapi import FastAPI, Request, Depends

from fastapi.responses import FileResponse

from pydantic import BaseModel

from typing import List, Dict, Optional

import uvicorn

import random

import uuid

import json

import pandas as pd

import torch

import torch.nn as nn

from transformers import AutoTokenizer, AutoModel

from sqlalchemy import create\_engine, Column, String, Float, Text, func

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker, Session

app = FastAPI()

# ======= Database Setup ========

DATABASE\_URL = "mysql+pymysql://username:password@localhost:3306/triage\_db"

engine = create\_engine(DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative\_base()

class PatientRecord(Base):

\_\_tablename\_\_ = "patient\_records"

record\_id = Column(String(64), primary\_key=True)

user\_id = Column(String(64))

symptoms = Column(Text)

predicted\_diseases = Column(Text)

recommended\_department = Column(String(128))

chat = Column(Text)

Base.metadata.create\_all(bind=engine)

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

# ======= Query & Statistics Endpoint ========

@app.get("/statistics")

def get\_statistics(db: Session = Depends(get\_db)):

results = db.query(PatientRecord.recommended\_department, func.count(PatientRecord.record\_id))\

.group\_by(PatientRecord.recommended\_department).all()

return {dept: count for dept, count in results}

@app.get("/patients")

def get\_all\_patients(db: Session = Depends(get\_db)):

records = db.query(PatientRecord).all()

return [

{

"record\_id": r.record\_id,

"user\_id": r.user\_id,

"symptoms": json.loads(r.symptoms),

"predicted\_diseases": json.loads(r.predicted\_diseases),

"recommended\_department": r.recommended\_department

}

for r in records

]

@app.get("/export\_excel")

def export\_excel(db: Session = Depends(get\_db)):

records = db.query(PatientRecord).all()

data = []

for r in records:

data.append({

"Record ID": r.record\_id,

"User ID": r.user\_id,

"Symptoms": ", ".join(json.loads(r.symptoms)),

"Diseases": ", ".join([f"{k} ({v})" for k, v in json.loads(r.predicted\_diseases).items()]),

"Department": r.recommended\_department

})

df = pd.DataFrame(data)

file\_path = "/mnt/data/patient\_records.xlsx"

df.to\_excel(file\_path, index=False)

return FileResponse(file\_path, filename="patient\_records.xlsx")